Characterization of Parallelism and Deadlocks in Distributed Digital Logic Simulation

Larry Soule and Anoop Gupta
Computer Systems Laboratory
Stanford University, CA 94305

Draft of November 3, 1988

Abstract

This paper explores the suitability of the Chandy-Misra algorithm for digital logic simulation. We use four realistic circuits as benchmarks for our analysis, with one of them being the vector-unit controller for the Titan supercomputer from Ardent. Our results show that the average number of logic elements available for concurrent execution ranges from 6.2 to 92 for the four circuits, with an overall average of 50. Although this is twice as much parallelism as that obtained by traditional event-driven algorithms, we feel it is still too low. One major factor limiting concurrency is the large number of global synchronization points — "deadlocks" in the Chandy-Misra terminology — that occur during execution. Towards the goal of reducing the number of deadlocks, the paper presents a classification of the types of deadlocks that occur during digital logic simulation. Four different types are identified and described both intuitively in terms of circuit structure and formally with equations. Using domain specific knowledge, the paper proposes methods for reducing these deadlock occurrences. For one of the benchmark circuits, the use of the proposed techniques eliminated *all* deadlocks and increased the average parallelism from 40 to 160. We believe that the use of such domain knowledge will make the Chandy-Misra algorithm significantly more effective than it would be in its generic form.

1 Introduction

Logic simulation is a very common and effective technique for verifying the behavior of digital designs before they are physically built. A thorough verification can reduce the number of expensive prototypes that are constructed and save vast amounts of debugging time. However, logic simulation is extremely time consuming for large designs where verification is needed the most. The result is that for large digital systems only partial simulation is done, and even then the CPU time required may be days or weeks. The use of parallel computers to run these logic simulations offers one promising solution to the problem.

Traditionally, the two commonly used parallel simulation algorithms for digital logic have been (i) compiled-mode simulations and (ii) centralized time event-driven simulations. In compiled-mode simulations, each logic element in the circuit is evaluated on each clock tick. The main advantage of this algorithm is its simplicity, the main disadvantage being that the processors do a lot of avoidable work, since typically only a small fraction of logic elements change state on any clock tick. The algorithm's simplicity makes it suitable for direct implementation in hardware [3,6], but such implementations make it difficult to incorporate user-defined models or represent the circuit elements at different levels of abstraction. In the second approach of centralized time event-driven algorithms, only those logic elements whose inputs have changed are evaluated on a clock tick. This avoids the redundant work done in the previous algorithm, however the notion of the global clock and synchronized advance of time for all elements in the circuit limits the amount of concurrency [2,14,17]. These centralized time approachs work efficiently on multiprocessors with 10 nodes or so [12,13,16], but for larger machines we need alternative approaches that move away from this centralized advance of the simulation clock.

The approach generating the most interest recently is the Bryant/Chandy-Misra distributed time discrete-event simulation algorithm [1,4,5,8,10,11,15]. It allows each logic element to have a local clock, and the elements communicate with each other using time-stamped messages. In this paper, we explore the suitability of the Chandy-Misra algorithm for parallel digital logic simulation. We use four realistic circuits as benchmarks for our analysis. In fact, one of the circuits is the vector-unit controller for the Titan supercomputer from Ardent. Our results show that

Submitted to DAC 1989.

the basic unoptimized Chandy-Misra algorithm results in an average *concurrency*[1] of 50 for the four circuits while being just as efficient as the event-driven algorithm. For two of the benchmark circuits, which were also studied in an earlier paper [14], the *unoptimized* Chandy-Misra algorithm extracted 40% and 107% more parallelism than the centralized time event-driven simulation algorithm.

The 50-fold average concurrency observed in the four benchmark circuits, however, is still too low. Once all the overheads are taken into account, the 50-fold concurrency may not result in much more than 10-20 fold speed-up. One major factor limiting concurrency is the large number of global synchronization points — "deadlocks" in the Chandy-Misra terminology — that occur during execution. We believe that understanding the nature of the deadlocks, why they occur and how their number can be reduced, is the key to getting increased concurrency from the Chandy-Misra algorithm. To this end, the paper presents a classification of the types of deadlocks that occur during digital logic simulation. Four different types are identified and described both intuitively in terms of circuit structure and formally with equations. Using domain specific knowledge, we then propose methods for reducing these deadlock occurrences. For one benchmark circuit, we show how using information about logic gates can eliminate all of the deadlocks. We believe that the use of such domain knowledge will make the Chandy-Misra algorithm significantly more effective than it would be in its generic form.

The organization of the rest of the paper is as follows. The next section describes the basic Chandy-Misra algorithm and some notation used in the paper. Next we describe the four benchmark circuits that were simulated to get the measurements. Section 4 presents measurements of the parallelism extracted by the algorithm and Section 5 presents the classification of the deadlocks and ways for resolving them. Finally, Section 6 presents a summary of the results and discusses directions for future research.

2 Background and Notation

2.1 Basic Chandy-Misra Algorithm, Deadlocks, and NULL Messages

We begin with a brief description of the basic Chandy-Misra algorithm [5] as applied to the domain of digital logic simulation. The simulated circuit consists of several circuit elements (transistors, gates, latches, etc) called *physical processes* (PP). One or more of these PPs can be combined into a *logical process* (LP), and it is with these LPs that the simulator works.[2] Each different type of LP has a corresponding section of code that simulates the underlying physical processes (note that the mapping between PPs and LPs is often trivial in gate-level circuits, with each gate represented as a simulation primitive). Each of these LPs has associated with it a *local* time that indicates how far the element has advanced in the simulation. Different LPs in the circuit can have different local times associated with them, and thus the name distributed time simulation algorithm. Each LP receives time-stamped event messages on its inputs and consumes the messages whenever all of the inputs are ready. As a result of consuming the messages, the logic element advances its local time and possibly sends out one or more time-stamped event messages on its outputs.

As an example, consider a two-input AND-gate with local-time 10, an event waiting on input-1 at time 20 (thus the value of input-1 is known between times 10 and 20), and no events pending on input-2. In this state, the AND-gate process is suspended and it waits for an event message on input-2. Now suppose that it gets an event on input-2 with a time-stamp of 15. The AND-gate now becomes active, consumes the event on input-2, advances its local time to 15, and possibly sends an output message with time stamp 15 plus AND-gate delay.

We now introduce the concepts of *deadlocks*. In the basic Chandy-Misra algorithm, even when input events are consumed and the local time of an LP is advanced, no messages are sent on an output line unless the value of that output changes. This optimization is similar to that used in normal sequential event-driven simulators where only elements whose inputs have changed are evaluated and it makes the basic Chandy-Misra algorithm just as efficient. However, this optimization also causes *deadlocks* in the Chandy-Misra algorithm. In a deadlock situation, no element can advance its local time, because each element has at least one input with no pending events. We reemphasize that this deadlock has nothing to do with a deadlock in the physical circuit, but it is purely a result of the optimization

[1] Note that, by concurrency we refer to the number of logic elements that could be evaluated in parallel if there were infinite processors.
[2] In this paper the terms LP and element are used interchangeably.

discussed above. The deadlock is resolved by scanning all the unprocessed events in the system, finding the minimum time-stamp associated with these events, and updating the input-time of all inputs with no events to this time (note that this deadlock resolution can also be done in parallel). Consequently, the basic Chandy-Misra algorithm cycles between two phases: the *compute* phase when elements are advancing their local time, and the *deadlock resolution* phase when all elements are stuck.

One way to totally bypass the deadlock problem is to not use the optimization discussed above. Thus elements would send output messages whenever input events are consumed and the local time of an element is advanced. This would be done even if the value on the output does not change. Such messages are called NULL messages in the Chandy-Misra terminology, as they carry only time information and no value information. Unfortunately, always sending NULL messages makes the Chandy-Misra algorithm so inefficient that it is not a good alternative to avoiding deadlocks. However, in this paper we show how *selective* use of NULL messages can significantly reduce the number of deadlocks that need to be processed.

Regarding parallel implementation of the Chandy-Misra algorithm, since each element is able to advance its local time *independently* of other elements, all elements can potentially execute concurrently. However, only when all inputs to an element become ready (have a pending event), is the element marked as available for execution, and placed on a distributed work queue. The processors take these elements off the distributed queue, execute them, update their outputs, and possibly activate other elements connected to the outputs. This happens until a deadlock is reached, when the deadlock resolution procedure is invoked.

2.2 Notation

As pointed out in the introduction, understanding the nature of deadlocks is key to increasing the parallel simulation performance. To help describe and understand the deadlocks, we now introduce some formal notation. Recall that each logical process has input and output event queues with time-stamped messages associated with it. For a particular LP_i, we have:

$\mathbf{E_{ij}}$ - the time of the earliest *unprocessed* event on input j of LP_i.

$\mathbf{E_i^{min}}$ - the minimum time of all the current input events of LP_i (short for $min_j E_{ij}$).

$\mathbf{V_i}$ - the maximum simulation time LP_i has progressed to.

$\mathbf{V_{ij}}$ - the simulation time the j^{th} input of LP_i is valid until.

$\mathbf{D_{ij}}$ - the propagation delay from any change in an input value to a change in the j^{th} output of LP_i.

$\mathbf{V_{ij}^O}$ - the simulation time the j^{th} output LP_i is valid until (usually $V_{ij}^O = V_i + D_{ij}$).

$\mathbf{O_{ij}}$ - the *node* connected to the j^{th} output of LP_i.

$\mathbf{I_{ij}}$ - the *node* connected to the j^{th} input of LP_i.

$\mathbf{C_{ij}}$ - Directed circuit connectivity: $\begin{cases} \text{True} & \text{if there is a link from } LP_i \text{ to } LP_j \\ \text{False} & \text{otherwise} \end{cases}$

In addition to the variables above, most circuits have some notion of a system clock and an associated cycle time, so let this cycle time be denoted as $\mathbf{T_{cycle}}$.

3 Benchmark Circuits

In this section, we first provide a brief description of the benchmark circuits used in our study and then some general statistics characterizing these circuits. The four circuits that we use are:

1. **Ardent-1:** This circuit is that of the vector control unit (VCU) for the Ardent Titan graphics supercomputer[7]. The VCU is implemented in a 1.5μ CMOS gate array technology and it provides the interface among the integer processing unit, the register file, and the memory. It also allows multiple scalar instructions to be executed concurrently by scoreboarding. It consists of approximately 45,000 two-input gates.

2. **H-FRISC:** A small RISC generated by the HERCULES [9] high-level synthesis system from the 1988 High Level Synthesis Workshop. The RISC instruction set is stack based and fairly simple. This circuit consists of approximately 11,000 two-input gates.

3. **Multiplier:** This circuit represents the inner core of a custom 3μ CMOS combinational 16x16 bit integer multiplier. Multiplies are pipelined and have a latency time of 70ns. The approximate complexity is 7,000 two-input gates.

4. **8080:** This circuit corresponds to a TTL board design that implements the 8080 instruction set. The design is pipelined, runs 8080 code at a speed of 3-5 MIPS, and provides an interface that is "pin-for-pin" compatible with the 8080. The approximate complexity is 3,000 two-input gates.

We note that the benchmark circuits cover a wide range of design styles and complexity — we have a large mixed-level synchronous gate array; a medium gate-level synthesized circuit; a medium gate-level combinational chip; and a small synchronous board-level design. The fact that we have both synchronous pipelined circuits and totally combinational circuits is also important, because they exhibit very different deadlock behavior during simulation.

We now present some general statistics for these benchmark circuits in Table 1. The statistics consist of:

- Element count: The number of primitive elements (LPs) in the circuit. One expects the amount of concurrency in the circuit to be positively correlated with this number (it is indeed so, as can be seen in Table 2).

- Element complexity: This is defined as the number of equivalent two input gates per primitive element. The number of primitive elements multiplied by the element complexity gives a more uniform measure for the circuit complexity. The element complexity also gives an indication of the compute time required to evaluate a primitive element, and thus specifies the grain of computation.

- Element fan-in/fan-out: The average number of inputs/outputs of an element. These numbers are also correlated to the element complexity. If the average number of inputs is high, one would expect a higher probability of deadlock as there are more ways in which one of the inputs may have no event.

- Percent logic and synchronous elements: The percentage of elements that are purely combinational logic and the percentage that have internal state. Pipelined designs like the Ardent and 8080, tend to have a higher percentage of synchronous elements.

- Net count: The number of wires in the circuit.

- Net fan-out: The average number of elements a wire is attached to. The Ardent and 8080 circuits have some global buses that affect many components. This fact is reflected in their high net fan-out numbers.

- Representation: The level of representation of the simulation primitives. A circuit made up of only logic gates and one-bit registers is at the gate-level while a design made up of TTL-like components is at the RTL-level.

Another important performance related aspect that we can infer from these numbers is the relative cost of resolving a deadlock. This cost of resolving a deadlock depends on the execution time of the models (related to element complexity) and the number of elements that must be checked and possibly activated. Thus we would expect deadlock resolution to be fairly cheap for the 8080 design with 281 elements, since there are so few elements to be checked and because each evaluation of an RTL element is much longer than a trivial logic operation. However, we would expect the relative cost of resolving a deadlock in the larger gate-level circuits (for example, H-FRISC) to be high due to the large number of components and the low execution time of the models.

Table 1: Basic Circuit Statistics

Statistic	Ardent-1	H-FRISC	Mult-16	8080
Element Count	13,349	8,076	4,990	281
Element Complexity	3.4	1.40	1.42	12
Element Fan-in	2.72	2.14	2.14	5.78
Element Fan-out	1.2	1.0	1.0	2.63
% Logic Elements	88.8	97.2	100	83.3
% Synchronous Elements	11.2	2.8	0.0	16.7
Net Count	13,873	8,093	5,077	748
Net Fan-out	2.66	2.14	2.14	5.48
Representation	gate/RTL	gate	gate	RTL
Basic Unit of Delay	0.5ns	unit	1ns	1ns

4 Parallelism Measurements

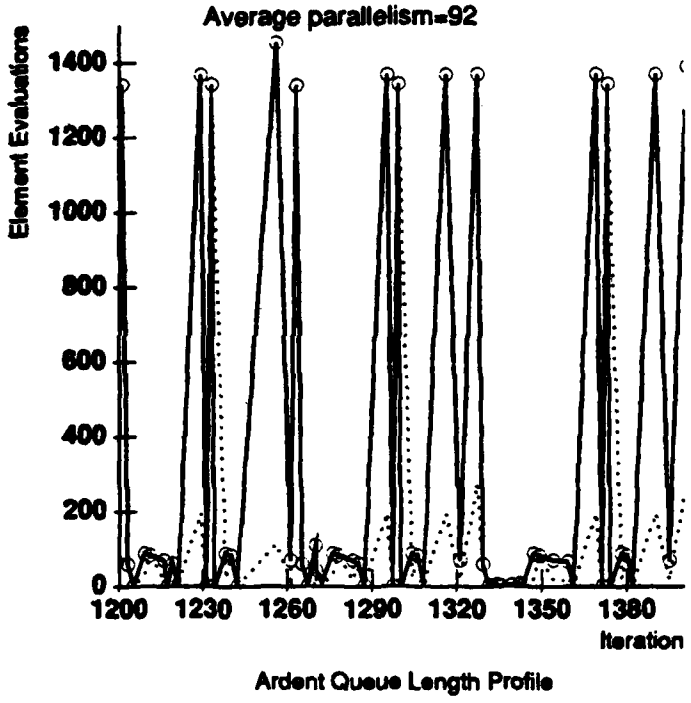
In this section, we discuss how parallelism is exploited by the Chandy-Misra algorithm and present data regarding the amount of concurrency available in the four benchmark circuits. We also present data regarding the granularity of computation, the number of deadlocks per clock cycle, and the amount of time spent in deadlock resolution. These numbers were gathered from our parallel implementation of the Chandy-Misra algorithm running on an Encore Multimax, a shared-memory multiprocessor with sixteen NS32032 processors, each processor delivering approximately 0.75 MIPS.

Since we are interested in the parallel implementations of the Chandy-Misra algorithm, the first question that arises is how much speed-up can be obtained if there were arbitrarily many processors, and if there were no synchronization or scheduling overheads. We call this measure the *concurrency* or intrinsic parallelism of the circuits under Chandy-Misra algorithm. For our concurrency data, we further assume that all element evaluations take exactly one unit of time. Thus, the simulation proceeds as follows. After a deadlock and the ensuing deadlock resolution phase, all elements that are activated (i.e., have at least one event on each of their inputs) are processed. This happens in exactly one unit-cost cycle as we assume arbitrarily many processors. The number of elements that are evaluated constitutes the concurrency for this *iteration*. The evaluation of the elements, of course, results in the activation of a whole new set of elements, and these are evaluated in one cycle in the next iteration. The computation proceeds on this way until a deadlock is reached, and we start all over again.

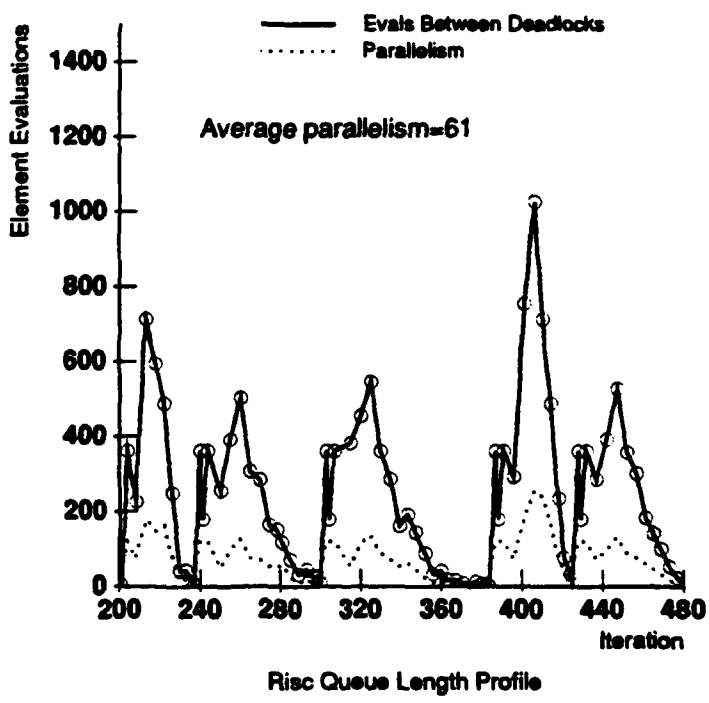
Figure 1 shows the concurrency data (shown using the dashed line) and event profiles (shown using the solid line) for the four benchmark circuits. The event profiles show a plot of the total number of logic elements evaluated between deadlocks. The profiles are generated over three to five simulated clock cycles in the middle of the simulation. We would like to reemphasize that the profiles in Figure 1 are not algorithm independent, but are specific to the basic Chandy-Misra algorithm. In fact, our research suggests enhancements to the basic Chandy-Misra algorithm, so that much more concurrency may be observed.

The profiles clearly show cyclical patterns with the highest peaks corresponding to the system clock(s) of the simulated circuits, and the portions between the peaks corresponding to the events propagating through the combinational logic between the sets of registers. The Ardent profile shows that the circuit quickly stabilizes after the clock with only a few deadlocks while the multiplier, with many levels of combinational logic, takes quite a while to stabilize with many deadlocks. This close correspondence between the event profiles and the circuit being simulated shows the importance of exploiting domain specific information; any circuit characteristic we change or exploit will be directly reflected in the event profiles. Understanding how these changes affect the profiles and being able to predict them is important in obtaining better performance. A summary of the concurrency information is also presented in Table 2. The top line of the table shows the concurrency as averaged over all iterations in the simulation.

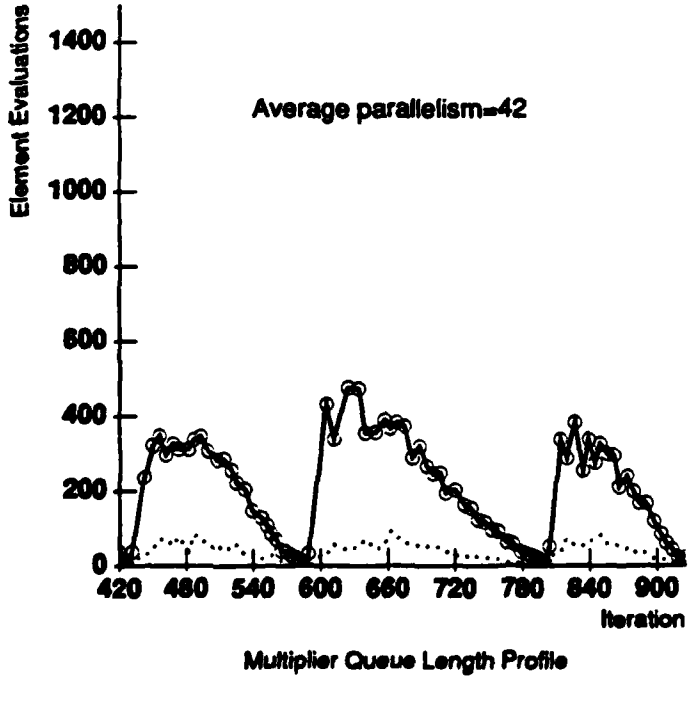
In addition to knowing how many concurrent element evaluations or tasks that are available, we also need to know



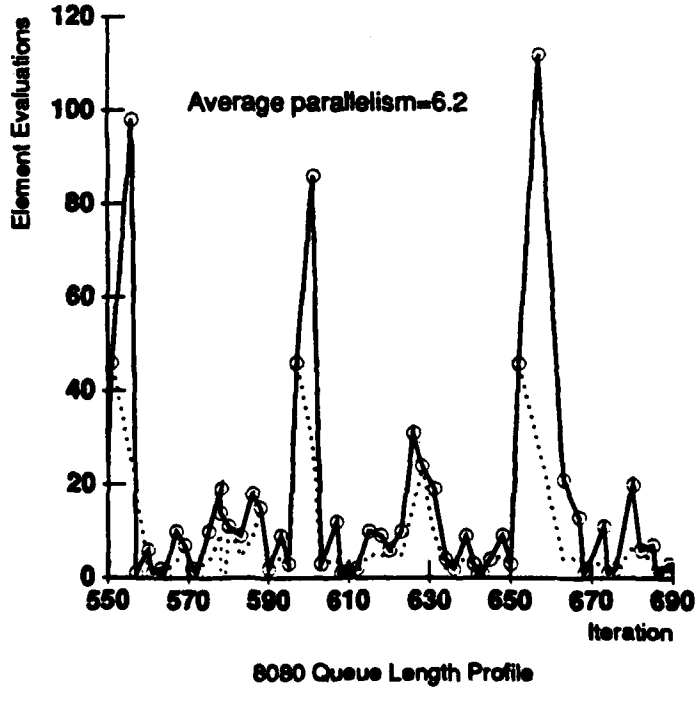
Ardent Queue Length Profile



Risc Queue Length Profile



Multiplier Queue Length Profile



8080 Queue Length Profile

Figure 1: Event Profiles

Table 2: Simulation Statistics

Statistic	Ardent-1	H-FRISC	Mult-16	8080
Unit-cost Parallelism	92	67	42	6.2
Granularity (ms)	0.74	0.66	0.75	2.61
Deadlock Ratio	308	245	248	15
Cycle Ratio	1,644	1,982	6,712	132
Deadlocks Per Cycle	5.3	8.1	27.1	8.9
Avg Deadlock Resolution Time (ms)	520	230	206	11
% Time in Deadlock Resolution	58	46	41	19

the task granularity and how often deadlocks (global processor synchronizations) occur. The granularity or basic task size for our application (a model evaluation) includes checking the input channel times, executing the model code, calculating the least next event and possibly activating the elements in its fan-out. The numbers discussing task granularity and frequency of deadlocks are summarized in Table 2. The table also presents the following ratios that help characterize the performance of the Chandy-Misra algorithm:

- **Deadlock ratio (DR):** Number of element evaluations divided by the number of deadlocks.
- **Cycle ratio (CR):** Number of element evaluations divided by the number of simulated clock cycles.
- **Deadlocks per cycle:** Number of deadlocks divided by the number of simulated clock cycles.

Since increased parallelism was the main motivation for using the Chandy-Misra algorithm, we now compare the concurrency it obtains to that obtained using a traditional event-based algorithm. For our comparison, we use the concurrency data presented for the 8080 and multiplier circuits in a parallel event-driven environment in [13,14]. These papers showed that the available concurrency was about 3 for the 8080 and 30 for the multiplier. From Table 2, the corresponding numbers for the Chandy-Misra algorithm are 6.2 for the 8080 and 42 for the multiplier. The fact that the concurrency increases only by a factor of 1.5-2 is somewhat disappointing, since Chandy-Misra algorithm is more complex to implement. However, we believe that using the techniques proposed in the next section, the Chandy-Misra algorithm can be suitably enhanced to show much higher concurrency.

The last two lines of Table 2 give data about the average time taken by each call to deadlock resolution and the total fraction of time spent in deadlock resolution. The cost of resolving a deadlock for the three larger circuits is indeed high, especially when compared to the cost of evaluating a logic element (see the granularity line). For example, in the time it takes to resolve a deadlock in Ardent, 700 logic element activations could have been processed. In H-FRISC, 350 elements could have been evaluated, and in the multiplier, 275 elements could have been evaluated. In our research, we are also exploring techniques to reduce the deadlock resolution time significantly by caching information from previous simulation runs of same circuit, but results are not available yet.

5 Characterizing Deadlocks

Even though there is reasonable parallelism available in the execution phase of the Chandy-Misra algorithm, deadlock resolution is so expensive in the larger circuits that it consumes 40-60% of the total execution time. Clearly we have to reduce this percentage in order to get good overall parallel performance. The first step towards this reduction is understanding why deadlocks occur and how they can be avoided. The types of deadlock that occur in logic simulation are characterized in this section and this characterization gives us insight into what aspects of logic simulation can be effectively exploited to achieve good overall performance.

In the logic simulations that were studied, the elements that became deadlocked can be put into two categories: (i) those deadlocked due to some aspect of the circuit structure (e.g topology, nature of registers, feed-back) and (ii)

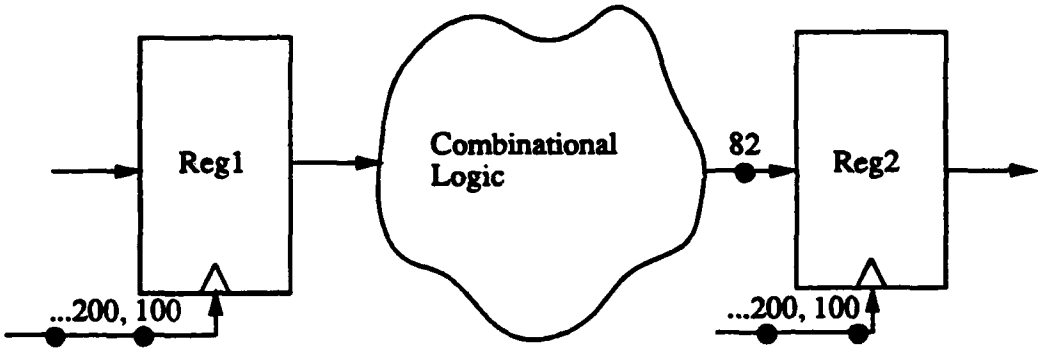


Figure 2: Deadlock Caused by a Clocked Register

those deadlocked due to low activity levels (e.g. typically only 0.1% of elements need to be evaluated on each time step in event-driven simulators[14]). In the following subsections, descriptions and examples of each of the types of deadlock are given, along with measurements that show how much each type contributes to the whole.

5.1 Registers and Generator Nodes

In a typical circuit, enough time is allowed for the changes in the output of one set of registers to propagate all the way to the next set of registers in the datapath and stabilize before the registers are clocked again. For example, in Figure 2, the critical path of the combinational part of the circuit is 82ns, and the clock node changes every 100ns to allow everything to stabilize. *Reg1* is clocked at the start of the simulation, and the events propagate through the combinational logic, generating an event at time 82. This event at time 82 is consumed by *Reg2* since the clock node is defined for all time in this example. However, the next event at time 100 is *not* consumed since the input to the latch is only defined up to time 82, not 100. This causes *Reg2* to block and the deadlock resolution phase is entered. This is a large source of deadlocks since most circuits have many registers, latches and generator nodes (e.g. clock(s), reset, inputs, etc.),

In Table 3 we see that for the Ardent, register-clock deadlocks account for 92% of all the elements activated in the deadlock resolution phase even though registers only make up 11% of the elements. This is mainly due to the pipelined nature of the Ardent design where there is only a small amount of combinational logic between register stages. In the case of the RISC design, there are more combinational logic between the registers than the Ardent and more logic gates connected to the input stimulus generators. Thus register-clock and generator deadlocks both cause around 20% of the deadlock activations for a total of 40%. In the multiplier design, there are many levels of logic between the inputs and outputs and does not have any registers. Thus there can not be any register-clock deadlocks and very few generator deadlocks. The 8080 design, like the Ardent, is pipelines and hence register-clock deadlocks are the main source of deadlock Here 55% of the activations are caused by register-clock deadlocks while only 17% of the elements are registers.

5.1.1 Detection

In order to measure how much any particular deadlock type affects the overall simulation, there must be some way of concretely identifying that type. A *register-clock deadlock* is said to occur whenever a clocked element LP_i that is activated during deadlock resolution has the earliest unprocessed event on its clock input. A *generator deadlock* is said to occur whenever the earliest unprocessed event was received directly from a generator element. In terms of the notation introduced earlier, this can be expressed as when $E_i^{min} \bmod T_{cycle} = 0$

Table 3: Register-Clock and Generator Deadlocks

Circuit	Total Deadlock Activations	Register-clock Activations	% of Total	Generator Activations	% of Total
Ardent-1	316.0k	290.0k	92	583	0.2
H-FRISC	45.6k	8.9k	20	8.800	19.0
Mult-16	27.2k	0.0k	0	40	0.1
8080	8.3k	4.6k	55	53	0.6

5.1.2 Proposed Solutions

Taking advantage of behavior: In general, an input event may arrive at any time in an element's future causing it to change its output. Thus, an element can only be sure of its outputs up to the minimum time its inputs are valid plus the output delay ($V_i + D_{ij}$). In the case of registers and latches, however, we know that the output will not change until the next event occurs on the clock input regardless of the other inputs. This knowledge of input *sensitization* is easy to use and potentially very effective since the outputs can be advanced up to the next clock cycle. In registers and latches with asynchronous inputs (like set, clear, etc.), those inputs must be taken into account as well as the clock node when determining the valid time of an output.

Fan-out Globbing: This technique reduces the overhead and the time needed to perform deadlock resolution. Recall that a particular LP is composed of many PPs. These PPs can be combined in different ways to form larger units. Combining many registers that share the same clock node will reduce the overhead of activating each register separately. Typically hundreds of one-bit registers and gates are connected to the clock node(s) and often times during deadlock resolution, the minimum event is on the clock node (as in the example above). If we combine these registers and gates in groups of n, we call this grouping *fan-out globbing with a clumping factor of n* since we are combining the fan-out elements of the clock. This reduces the overhead of inserting and deleting the elements in the evaluation queue. However, since it combines elements, it also reduces the parallelism available. We are currently looking into just how much reduction in overhead and parallelism this causes.

5.2 Multiple Input Paths with Different Delays

Whenever there are multiple paths with different delays from a node to an element, there is a chance of that element deadlocking. An example of this is the MUX shown in Figure 3. There are two paths from the Select line to the OR-gate at the output. If the Data and ScanData lines are valid, an event on the Select node could propagate through the two paths and generate events at times 11 and 12. The event at time 12 will not be consumed by the OR-gate since its other input is only defined up to time 11 causing the OR-gate to deadlock. Thus, multiple paths from a node to an element can result in an unconsumed event on the path with the larger delay.

5.2.1 Detection

Let LP_i be the deadlocked element and j be the index of the input with the unprocessed event (i.e. j such that $E_{ij} = E_i^{min}$). Then if there are two different paths from some element, LP_k, to the deadlocked element, LP_i, with the longer path ending at input j, then a *multiple path* deadlock has occurred.

5.2.2 Proposed Solutions

Since this type of deadlock is due to the local topology of the circuit, there is no easy way of avoiding it. However, there are a couple of options.

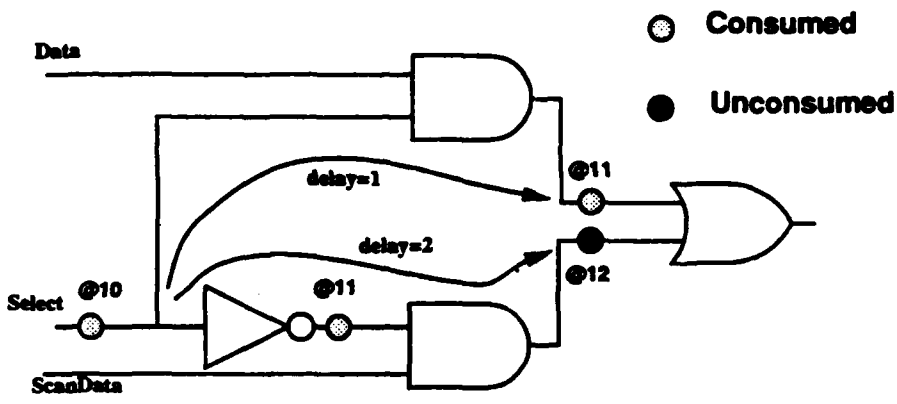


Figure 3: Deadlock due to Multiple Paths of Different Delays

Demand-driven: The elements that are affected by multiple paths could be marked either while compiling the netlist or from previous simulation runs. When these elements are executed, a demand driven technique could be used. With a demand-driven technique, whenever an element can not consume an input event, requests are made to its fan-in elements (the ones driving its input pins) asking "Can I proceed to this time?". These requests propagate backwards until a yes or no answer can be ensured. Propagating these requests can be expensive especially if there are long feedback chains in the circuit. Thus we must be very selective in the elements we choose to use this technique with.

Structure globbing: If there are not too many elements involved in the multiple paths, we may be able to *hide* the multiple paths by globbing those elements into one larger *LP*. However, the composite behavior of the gates must be generated and the detailed timing information must be preserved. Preserving the exact timing information is non-trivial. In essence a state variable must be made for each of the internal nodes and the element may have to schedule itself to make the outputs change at the correct times. This self-scheduling may cause the element to deadlock because, by requesting itself to be evaluated at some time, it must wait until the inputs are valid up to that time just as before. If the detailed timing information does not need to be preserved, the composite behavior is easy to generate (compiled-code simulation techniques can be used on the small portion of the circuit that is being globbed together) and this deadlock type will be avoided.

Taking advantage of behavior: If we know the behavior of an element, it may be possible to advance that element even though some of its inputs are not known. For example in Figure 3, if the event at time 11 going into the OR-gate has a value of 1, the output is known to be 1 regardless of the value of the other input and the OR-gate need not deadlock. In a gate-level simulation, the behavior of most of the elements is very simple and can be readily exploited.

5.3 Order Of Node Updates

The *activation criteria* for the basic Chandy-Misra algorithm is: activate an element only when an event is received on one of its inputs. Sometimes this activation criteria can cause a consumable input event to be stranded due to the order in which the node updates are performed. This stranded event will cause the element to deadlock. In Figure 4, element e1 consumes the event at time 10, produces an event at time 11, and activates element e3. If e3 is now executed, e3 will not be able to consume the event at time 11 because the input from e2 will not be valid at time 11. If an event at time 10 now arrives at e2 and e2 is evaluated, it will update the valid-time of the input to e3 but it will not activate e3 because no *event* was generated. If e3 had waited for e2 to be evaluated, the inputs to e3 would have both been valid at time 12 and the event at time 11 could have been consumed.

In Table 4, we see that the order of node updates type of deadlock is uncommon in the Ardent simulation which is

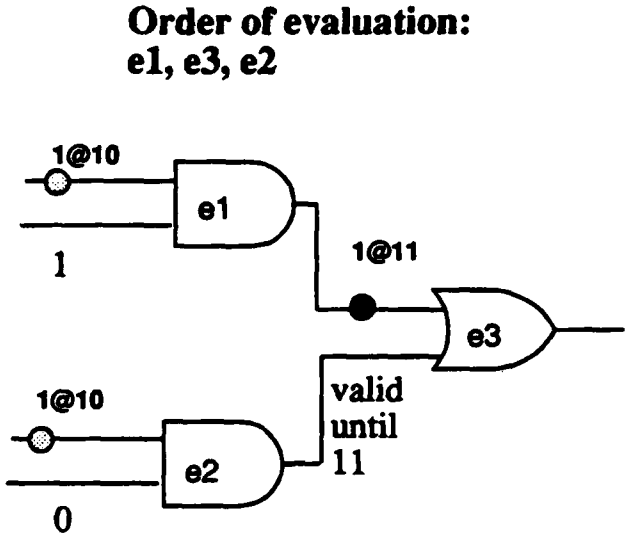


Figure 4: Deadlock Caused by Order Of Node Updates

Table 4: Deadlock Activations Caused by the Order of Node Updates

Circuit	Deadlock Activations	Order of Node Updates	% of Total
Ardent-1	316.0k	1.4k	0.4
H-FRISC	45.6k	1.0k	2.2
Mult-16	27.2k	1.7k	6.2
8080	8.3k	0.2k	2.2

dominated by the register-clock deadlocks. The order of node updates is, however, important in the combinational multiplier with its many levels of logic

5.3.1 Detection

Suppose LP_i is activated during deadlock resolution because it was waiting to consume an event at time t. If *all* of the input nodes are found to have advanced up to time t — that is if the element can safely consume the event without *any* input times being updated, an *order of node updates* deadlock has occurred. In the notation introduced earlier, this happens if $\min_j V_{ij} \geq E_i^{min}$

5.3.2 Proposed Solutions

New activation criteria: The problem is that the activation criteria does not activate an element when the valid times of its input node are updated. The problem can be eliminated if an element checks its fan-out elements when it updates the time of its output nodes. Any of those fan-out elements that have a real event at a time less than or equal to the new valid-time, should be activated. In the example, e2 would activate e3 when it updated the valid-time of its output to 11 since e3 has a real event at time 11. Note that this only works for the case where the updated node is directly connected to the element with the unconsumed event. If there are any intermediate

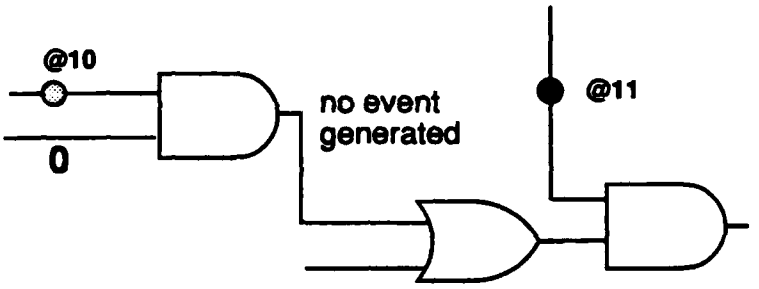


Figure 5: Deadlock Caused by Unevaluated Path

elements the deadlock is considered to be caused by an unevaluated path which is explained in the next subsection. If e3 had a third input, it still may not be able to consume the event at time 11 even after e2 is evaluated. This extra activation creates needless work and the effectiveness of this solution depends on the relative cost of performing a deadlock resolution on the particular circuit being simulated.

We can describe this new activation criteria formally by doing the following after each LP_i is evaluated:

For each output j
  For each LP_k connected to output j
    If $V_{ij}^{O} \geq E_k^{min}$
      then Activate LP_k

Rank ordering: The *rank* of an element is the maximum number of levels of logic between the element and any registers. It can be computed by assigning all registers and generator elements a rank of 0 and then iterating through the combinational elements assigning them a rank of one plus the maximum rank of its input elements. If the elements in the evaluation queue are ordered by their rank, the node updates will proceed in a more ordered fashion (i.e. elements farther away from the registers and external inputs that affect it will be evaluated later possibly letting their inputs become defined). In the example, e2 would be inserted before e3 since the inputs to e3 depend on the outputs of e2.

Since the rank information is easy to compute while compiling the netlist, the *run-time* cost is very little. Also, this technique does not generate any extra activations so the overall cost is cheap.

5.4 Unevaluated Path

The elements in the fan-out of a wire are activated only when a real event is produced on that wire. Thus, if element LP_i consumes an event but does not produce a new event (i.e. the activation does not result in a change in value of output signals), all paths from LP_i to the other elements will not be evaluated or updated. Figure 5 shows the case where one event is consumed and, since no new event is produced, the OR-gate is not activated and the second AND-gate can not consume the event at time 11 since the valid-time of the input from the OR-gate was not updated.

In Table 5 we see that unevaluated paths are very important in three of the four circuits. This is especially true for the RISC and multiplier designs which consist of many levels of combinational elements. For the RISC, the number of deadlocks caused by unevaluated paths is around 60% and that for the multiplier around 90%. In contrast, unevaluated paths are relatively unimportant in simulations of the pipelined Ardent design.

Table 5: Deadlock Activations Caused by Unevaluated Paths

Circuit	Deadlock Activations	One Level NULL	% of Total	Two Level NULL	% of Total	Combined %
Ardent-1	316.0k	3.0k	1.0	21.0k	6.6	8
H-FRISC	45.6k	4.3k	9.4	22.6k	49.6	59
Mult-16	27.2k	1.5k	5.5	23.8k	87.5	93
8080	8.3k	0.5k	5.7	2.9k	34.9	41

5.4.1 Detection

If NULL messages are *always* sent, the simulation will never deadlock (see Section 2.1). Unfortunately, this is highly inefficient since typical activity levels in event-driven simulators are around 0.1% in each time step. To find out how many deadlocks we could avoid by only *selectively* sending NULL messages, we did the following. We measured how many deadlock activations would have been avoided if every deadlocked element had received NULL messages from its immediate fan-in — corresponding to what we call "one level" of NULL messages — and how many activations would have been avoided by two levels of NULL messages.

To define this more concretely, let the *distance* between LP_i and LP_j be the minimum number of elements, $(e_1, e_2, ...e_k)$ such that $C_{ie_1}, C_{e_1e_2}, ..., C_{e_kj}$ are all true. Let this distance be denoted by δ_{ij} and the minimum delay between LP_i and LP_j by τ_{ij}. Using these definitions we get: LP_i was deadlocked by an unevaluated path of n levels,

If For each input j where $V_{ij} < E_i^{min}$
and each LP_k where $\delta_{ki}=n$ and the path ends at input j
$(V_k + \tau_{ki}) \geq E_i^{min}$ holds

5.4.2 Proposed Solutions

Caching: Since the activity levels are so low, we need to be very selective about which elements should send NULL messages. The proposed selection process follows the concept of *caching*. By caching information from previous runs, we can identify the elements that repeatedly deadlock due to an unevaluated path as the simulation progresses. When these elements get activated, they will send out NULL messages whenever their outputs times advance. In order to be effective, the caching algorithm must be quick and effective.

Taking advantage of behavior: If we know the behavior of an element, it may be possible to advance that element even though some of its inputs are not known. For example, if the event at time 11 going into the AND-gate of Figure 5 has a value of 0, the output is known to be 0 regardless of the value of the other input. In a gate-level simulation, the behavior of most of the elements is very simple and can be readily exploited. As it turns out, this technique works very well for the combinational multiplier circuit. It eliminates *all* deadlocks and increases the parallelism from 40 to 160.

5.5 Summary of the Contributions from each Deadlock Type

A summary of the composition of an average deadlock for the benchmark circuits is given in Table 6. In all but the Ardent circuit, the main deadlock type is the two-level NULL caused by unevaluated paths which are, in turn, caused by the very low activity levels in digital logic simulations. The Ardent and 8080 deadlocks are made up predominantly of register-clock deadlocks. They account for 92% and 55% of the deadlock activations even though synchronous elements comprise only 11 to 17% of the total elements. This is mainly due to the heavily pipelined

Table 6: Deadlock Activations Classified by Type

Circuit	Total Deadlock Activations	Register-clock Activations	Generator Activations	Order of Node Updates	One Level NULL	Two Level NULL
Ardent-1	316.0k	290.0k	583	1.4k	3.0k	21.0k
RISC	45.6k	8.9k	8,800	1.0k	4.3k	22.6k
Mult	27.2k	0.0k	40	1.7k	1.5k	23.8k
8080	8.3k	4.6k	53	0.2k	0.5k	2.9k

nature of the two circuits — lots of latches with only a few levels of logic in between. Thus most of the deadlocks occur when the registers and latches are waiting for their inputs to become valid.

The main contributors to deadlock in the RISC circuit (after the two-level NULL deadlocks), are generator and register-clock deadlocks. This is due to the consistent control style used by the synthesis system. The system clocks are generated externally and first pass through a level of logic that controls which parts of the design are active. These qualified clocks are then distributed to their corresponding circuit sections — the result being that most registers are waiting on their inputs and the elements connected to the generator nodes are waiting on their other inputs.

The multiplier design is highly interconnected with many levels of logic. Almost all of the deadlock activations are caused by the unevaluated paths in the circuit as shown by the two-level NULL column. This is caused by a few paths that are active all the way from the inputs to the outputs while most of the paths do not have any activity at all after the first couple of levels.

6 Conclusions

In characterizing the parallelism in distributed-time simulations of real circuits, we have shown that the Chandy-Misra algorithm extracts an average parallelism of 50 for the four benchmark circuits used. While this is 1.5-2 times better than traditional parallel event-driven algorithms, it is still too low to be used effectively in large parallel processing systems. Since deadlocks are the major factor limiting parallelism and the overall performance, the paper focused on understanding the nature of deadlocks. We classify the deadlocks that occur in logic simulation into four types: register clocks and generator nodes, multiple paths, unevaluated paths and the order of node updates. These four types are able to cover almost all of the deadlocks that occur. Concentrating on each type, we presented specific solutions to avoid or resolve the deadlocks caused by that type. Preliminary results show that we can eliminate all of the deadlocks in the multiplier simulation raising the parallelism from 40 to 160. These solutions exploit several different aspects of circuit behavior and we feel that it is with this domain specific knowledge that significantly better parallel performance can be achieved.

7 Acknowledgements

The authors are supported by DARPA contract N00014-87-K-0828. In addition, Anoop Gupta is supported by a DEC faculty award, and Larry Soule is supported by DEC, and by a National Science Foundation Graduate Fellowship.

References

[1] Marc Abrams. The Object Library for Parallel Simulation (OLPS). In *Winter Simulation Conference, 1988*, December 1988.

[2] M. Bailey and L. Snyder. An Empirical Study of On-Chip Parallelism. In *25th Design Automation Conference*, pages 160–165. University of Washington, June 1988.

[3] Tom Blank. A survey of hardware accelerators used in computer-aided design. *IEEE Transactions on Design and Test*. 21–39, August 1984.

[4] R. E. Bryant. *Simulation of Packet Communication Architecture Computer Systems.* Technical Report MIT.LCS.TR-188. MIT. July 1977.

[5] K. M. Chandy and J. Misra. Asynchronous Distributed Simulation Via a Sequence of Parallel Computations. *Comm of the ACM*, 24(11):198–206, April 1981.

[6] Monty Denneau. The Yorktown Simulation Engine. In *19th Design Automation Conference*, page 7.2. ACM/IEEE. 1982.

[7] Tom Diede, Carl Hagenmaier, Glen Miranker, Johnathan Rubinstein, and William Worley. The Titan Graphics Supercomputer Architecture. *Computer*, 21(9):13–30. September 1988.

[8] Richard Fujimoto. Lookahead in Parallel Discrete Event Simulation. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 34–41, University of Utah, 1988.

[9] David Ku and Giovanni DeMicheli. HERCULES - A System for High-Level Synthesis. In *25th Design Automation Conference*. pages 483–488, ACM/IEEE, June 1988.

[10] David Nicol. Parallel Discrete-Event Simulation of FCFS Stochastic Queueing Networks. In *PPEALS 88*, pages 124–137, ACM, 1988.

[11] D. Reed, A. Maloney, and B. McCredie. Parallel Discrete Event Simulation: A Shared Memory Approach. *IEEE Transactions on Software Engineering*, 14(4):541–553, April 1988.

[12] J. Smith, K. Smith, and R. Smith. Faster Architectural Simulation Through Parallelism. In *24th Design Automation Conference*, pages 189–194, ACM/IEEE, June 1987.

[13] Larry Soule and Tom Blank. Parallel Logic Simulation on General Purpose Machines. In *Proceedings of the 25th Design Automation Conference*, pages 166–171, Stanford University, 1988.

[14] Larry Soule and Tom Blank. Statistics for Parallelism and Abstraction Level in Digital Simulation. In *Proceedings of the 24th Design Automation Conference*, pages 588–591, Stanford University, 1987.

[15] David Wagner, Edward Lazowska, and Brian Bershad. *Techniques for Efficient Shared-Memory Parallel Simulation.* Technical Report 88-04-05, University of Washington, Department of Computer Science, August 1988.

[16] Andrew Wilson. *Parallelization of an Event Driven Simulator on the Encore Multimax.* Technical Report ETR 86-005, Encore Computer, 1986.

[17] Franklin Wong. Statistics on Logic Simulation. In *23rd Design Automation Conference*, pages 13–19, ACM/IEEE, July 1986.